UNIVERSIDADE FEDERAL DE MINAS GERAIS
Departamento de Ciência da Computação

Disciplina	Turmas
Algoritmos e Estruturas de Dados II	
Professores	

Data de entrega: 30/06/2016 **até as** 23:55 **via** ***Moodle***

Trabalho Prático em Linguagem C (TP2)

Especificação

O objetivo deste trabalho é implementar e aplicar os conhecimentos aprendidos em sala de aula sobre tabelas Hash, árvores binários e árvores SBB. Para isto deverá ser criado um programa que recebe dados de aluno como entrada (matrícula e nome) e os distribui por uma tabela Hash que resolve colisão utilizando árvore binária.

Detalhes de implementação

Dado os TADs a seguir, apresente uma implementação de tabela Hash que resolve colisão utilizando árvore binária sem balanceamento e árvore SBB.

- **Aluno:**

```
typedef int Chave;

struct Aluno
{
    char* nome;
    Chave matricula;
};

Aluno* criaAluno(char* n, Chave mat); // cria Aluno dado seu nome e sua matricula
void apagaAluno(Aluno* a); // desaloca ponteiro aluno (e seus dados)
void imprimeAluno(Aluno* a); // imprime dados de Aluno no formato: (matricula) nome
```

- **Árvore binária:**

```
typedef Aluno Elemento;

struct Arvore {
    Arvore* esq;
    Arvore* dir;
    Elemento* reg;
};
```

```
Arvore* criaArvore(Elemento* r); //cria uma árvore com o elemento r sendo raíz
Elemento* pesquisa(Arvore *t, Chave x); //pesquisa na árvore o elemento com chave x
void insereElemento(Arvore* t, Elemento *n); //insere elemento n na árvore t
Arvore* remove(Arvore* t, Chave c); //remove o elemento de chave c da árvore t
Arvore* achaMenor(Arvore* t); //encontra o menor elemento da árvore t
void imprimeArvore(Arvore *t); //imprime a ávore (pré-ordem, in-ordem ou pós-ordem)
void apagaArvore(Arvore *t); // desaloca arvore t
```

- **Árvore SBB:**

```
typedef Aluno Elemento;

struct ArvoreSBB {
Elemento* reg;
struct ArvoreSBB* esq;
struct ArvoreSBB* dir;
int esqtipo;
int dirtipo;
};

ArvoreSBB* criaArvoreSBB(Elemento* r); //cria uma árvore com o elemento r sendo raíz
Elemento* pesquisa(ArvoreSBB *t, Chave x); //pesquisa na árvore o elemento com chave x

void ee(ArvoreSBB** ptr);
void dd(ArvoreSBB** ptr);
void ed(ArvoreSBB** ptr);
void de(ArvoreSBB** ptr);
void iInsere(Elemento* r, ArvoreSBB** ptr, int *incli, int *fim);
void iInsereAqui(Elemento* r, ArvoreSBB** ptr, int *incli, int *fim);
void insereElemento(ArvoreSBB** t, Elemento *n); //insere elemento n na árvore t
void inicializa(ArvoreSBB** raiz);
void imprimeArvore(ArvoreSBB* t); //imprime a ávore (pré-ordem, in-ordem ou pós-ordem)
void apagaArvore(ArvoreSBB *t); //desaloca arvore t
```

- **Tabela Hash:**

```
struct Hash
{
    Arvore** hash;
    ArvoreSBB** hashSBB;
    int tam;
    int nElem;
};

int funcaoHash(...); //função Hash (deve criar como desejar, com a passagem de parâmetros que ju
Hash* criaHash(); //inicializacao padrao
Hash* criaHash(int t); //inicializa com tamanho passado (aloca)
void apagaHash(Hash* h); //desaloca Hash h
void insereNaHash(Hash* h, Elemento* x); //insere um Elemento na Hash
Elemento * obtemDaHash(Hash* h, Chave c); //obtem um Elemento da Hash dada sua chave
void imprime(Hash* h); //Imprime a tabela Hash h
int obtemNumElem(Hash* h);
```

Saída

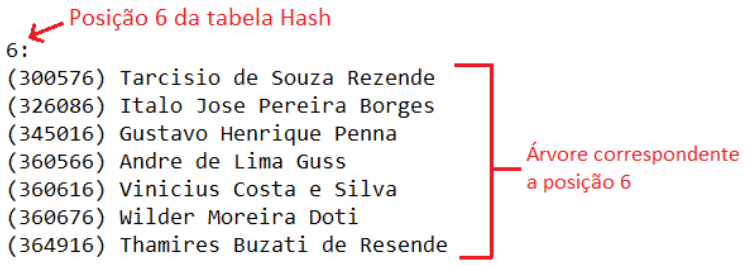
Seu programa deve escrever a solução em um arquivo no seguinte formato:

```
0:
(9300) Alessandro Gomes Sifuentes
(326080) Ewerton Ozorio da Luz
(331630) Thiago Paiva Medeiros
(360580) Diego Freitas Siqueira Souza
1:
(312211) Michael Bruno Pereira de Castilho
(360591) Gustavo Lopez Antunes Rezende
(369141) Rafael Silva Correia
2:
(355902) Tatianne Tinel Peixoto
3:
(277683) Joao Gabriel Nascentes Fernandes
(345023) Raissa do Vale Azevedo
(360573) Carlos Alberto de Oliveira
(360583) Felipe Pena Aguiar Marques
(360603) Matheus Fialho Anastacio
4:
(356044) Lucio Jose Pimenta Neto
(360594) Igor Antenor Leal
(360604) Nathan de Oliveira Resende
5:
(91085) Marcos Rodrigues Timo
(360565) Anderson de Souza Baptista de Leo
(360585) Filipe da Costa Matos Sousa Faria
(368545) Tiago Leocadio da Silva
6:
(300576) Tarcisio de Souza Rezende
(326086) Italo Jose Pereira Borges Oliveira
(345016) Gustavo Henrique Penna Duarte
(360566) Andre de Lima Guss
(360616) Vinicius Costa e Silva
(360676) Wilder Moreira Doti
(364916) Thamires Buzati de Resende
7:
(316487) Yuri Ribeiro Almeida(360607) Rafael Rodrigues Lima
(364897) Paula Ribeiro Rezende
8:
(369148) Andre Souza Espirito Santo
9:
(355899) Rafael Patricio de Souza
(360589) Guilherme Caires de Miranda
```

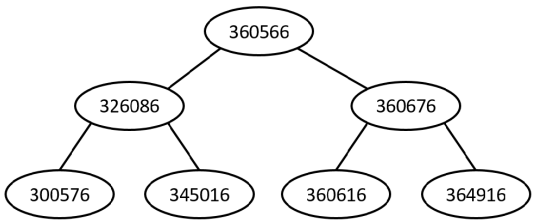
onde cada número representa a posição da tabela Hash e logo abaixo dele a árvore correspondente aquela posição impressa em "in-ordem" (caminhamento central). Mais detalhes na Figura 1.

O que deve ser feito

1. Implementar uma tabela Hash que resolva problemas de colisão com árvore binária sem balanceamento.



(a) Posição 6 da tabela Hash.



(b) Ilustração da árvore correspondente.

Figura 1: Posição 6 da tabela Hash e sua árvore correspondente ("in-ordem")

2. Implementar uma tabela Hash que resolva problemas de colisão com árvore binária SBB.
3. Fazer um comparativo de tempo entre as duas abordagens.
4. Fazer uma comparação entre as ordens de complexidade das duas abordagens
5. Todas as funções implementadas devem possuir um cabeçalho conforme o exemplo a seguir:

```
/*-------------------------------------------------------------------
Protótipo: Elemento* obtemDaHash(Hash* h, Chave c)
Função: Realiza a operação ...
Entrada: estrutura Hash contendo ...
Saída: Retorna o elemento ...
-------------------------------------------------------------------*/
```

6. O nome do arquivo de entrada e saída deverão ser passados por argumentos para o programa.

   ```
   ./exec entrada.txt saida.txt
   ```

   onde *exec* é o nome do executável, *entrada.txt* representa o nome do arquivo de entrada (por exemplo, alunos01.txt) e *saida.txt* é o nome do arquivo de saída (por exemplo, saidaAlunos01.txt).

7. Seu programa não deverá ter nenhum *leak* de memória, ou seja, tudo que for alocado de forma dinâmica, deverá ser desalocado com a função "free()" antes do término da execução.
8. O programa deverá ser possível de ser compilado no Linux, portanto ele não deverá conter nenhuma biblioteca que seja específica do sistema operacional Windows.
9. Você deverá implementar o trabalho na IDE Code::Blocks (*disponível em: http://www.codeblocks.org/*). Deve ser submetido ao moodle o diretório do projeto criado no Code::Blocks em um arquivo compactado contendo os arquivos ".h" (com as declarações das estruturas e das funções) e ".c" (um com as implementações das funções descritas nesse documento e outro com a função *main*). Além disso, a submissão também deverá conter o relatório do trabalho em formato PDF.
10. O trabalho deverá ser entregue via *Moodle* até as 23:55 horas do dia 30/06/2016. **Trabalhos que forem entregues fora do prazo não serão aceitos em nenhuma hipótese.**

Exemplos para teste

Para avaliar o funcionamento do seu programa disponibilizamos os arquivos *alunos01.txt*, *alunos02.txt* e *alunos03.txt*. Os arquivos são composto pelo formato "número de matrícula" (linha 1) e "nome do aluno" (linha 2), como ilustrado abaixo:

```
009300
Alessandro Gomes Sifuentes
360565
Andre de Lima Guss
360573
Carlos Alberto de Oliveira
360580
Diego Freitas Siqueira Souza
326080
Ewerton Ozorio da Luz
360583
Filipe da Costa Matos Sousa Faria
360589
Guilherme Caires de Miranda
345016
...
```

Documentação

Escreva um documento explicando o seu código e avaliando o desempenho de sua implementação. Separe-o em cinco seções: introdução, implementação, resultados, conclusão e referências. Seja claro e objetivo.

- **Introdução**: Faça uma breve introdução o problema, definindo-o com as *suas* palavras.
- **Implementação**: Explique quais foram as estratégias adotadas para a leitura dos arquivos, execução do programa e estruturas de dados utilizadas. Descreva qual o papel de cada função, seus parâmetros de entrada e de saída. **Faça a análise da sua implementação em termos de tempo e espaço indicando as funções e as ordens de complexidade de cada um. Responda as perguntas: o seu algoritmo tem complexidade polinomial ou exponencial? Por que? Você consegue pensar em alguma forma de resolver o problema com um custo mais baixo do que a proposta passada neste documento? (Hash + Árvores)**
- **Resultados**: Faça testes com seu programa utilizando várias arquivos, **inclusive alguns além dos passados nos exemplos.** Apresente os resultados obtidos e faça uma breve discussão sobre eles. Serão disponibilizados três arquivos para teste.
- **Conclusão**: Explique quais foram as dificuldades encontradas durante o desenvolvimento e as conclusões obtidas.
- **Referências**: Se você utilizou informações adicionais além das especificadas neste documento, cite as fontes.

Avaliação

- **4 pontos** pela implementação, onde serão avaliados, além do funcionamento adequado do programa: identação correta do código, comentários das funções, alocações de desalocações dinãmicas bem feitas e modularização.
- **4 pontos** pelos testes, onde serão avaliados os resultados obtidos.
- **6 pontos** pela documentação, onde serão avaliados a clareza das seções e a discussão dos resultados obtidos.
- ***Lembramos que será utilizado um software de detecção de cópias e qualquer tipo de plágio detectado resultará em nota 0 para ambos trabalhos!***

Pontos Extras

Todos os alunos da UFMG são bons programadores, mas sabemos que alguns destes alunos se destacam entre os demais. Para que estes alunos não fiquem entediados após terminarem o TP, vamos distribuir 1 (um) ponto extra para o aluno que criar a melhor função Hash (com menos colisões). Para isto considere uma tabela Hash de tamanho 60. **Nota.** Você precisa ter feito todo o TP para poder receber os pontos extras!